Full Stack Development with MERN

1.Introduction

- **Project Title :** Resolve Now: Your Platform For Online Complaints
- **Team Members :**

S.No	Name of the Member	Role
1.	Md. Manntasha Aara	Backend Development DataBase Development
2.	M. Priyanka	Frontend Development DataBase Development
3.	M.Varun Kumar	Project Setup and Configuration
4.	M.Vamsi	Project Implementation

2.Project Overview

- **Purpose :**

The primary purpose of *ResolveNow* is to provide a centralized, secure, and user-friendly platform for individuals and organizations to submit, track, and resolve complaints related to services or products. By digitizing and automating the complaint handling process, the platform improves transparency, responsiveness, and customer satisfaction. It acts as a bridge between users and customer service agents, facilitating smooth and accountable resolution workflows.

Goals :

- **Streamline Complaint Handling:**
  Automate the submission, assignment, tracking, and resolution of complaints to reduce manual intervention and delays.
- **Enhance User Experience:**
  Provide an intuitive interface where users can easily register, log in, file complaints, and communicate with agents in real time.
- **Ensure Real-Time Tracking and Notifications:**
  Keep users informed about their complaint status via dashboard updates and automated email/SMS alerts.
- **Facilitate Agent Interaction:**
  Enable direct communication between users and assigned agents to clarify issues, provide updates, and ensure quick resolutions.
- **Optimize Resource Allocation:**
  Use intelligent routing mechanisms to assign complaints to the most suitable agents or departments based on type and complexity.

- **Maintain Data Security and Confidentiality:**
  Protect sensitive user information through strong authentication, access controls, and encrypted data transmission and storage.
- **Empower Administrators:**
  Allow admin users to monitor all complaints, manage user roles, and ensure policy and compliance adherence across the system.
- **Enable Feedback and Quality Improvement:**
  Collect user feedback on the resolution process to improve service quality and agent performance.

- **Integrate Scalable Technology Stack:**
  Leverage technologies like MERN stack, Socket.IO, and WebRTC to support real-time data flow and seamless communication in a scalable architecture.
- **Promote Accountability and Transparency:**
  Maintain a full audit trail of interactions and complaint statuses to ensure transparent operations and resolve disputes effectively.

- **Features**

- **User Registration and Authentication**

  Secure sign-up and login system with email verification.

  Role-based access control for users, agents, and admins.

  Password encryption for data protection.

- **Complaint Submission**

  Users can register complaints by providing details such as:

    Complaint title & description

    Category/type of issue

    Contact information

    Attachments (images, documents)

  Real-time form validation for smooth user experience.

- **Complaint Tracking Dashboard**

    Users can view all their submitted complaints.

    Real-time status updates: *Pending*, *In Progress*, *Resolved*.

    Timeline of actions taken on each complaint.

- **Notifications System**

    Complaint submission confirmation

    Assignment to agent

    Resolution updates

    Agent responses and chat messages

- **Agent-User Messaging**

    Integrated chat system between user and assigned agent.

    Enables discussion, clarification, and resolution tracking.

Chat logs stored for transparency.

- **Agent Complaint Management**

  Agents can:

  View assigned complaints

  Update status and progress

  Communicate with users

- **Security and Data Confidentiality**

  User authentication and role-based access

  Encrypted data storage (MongoDB)

  Secure APIs using HTTPS and access tokens

- **Admin Dashboard**

  Monitor overall platform activity

  Assign complaints manually if needed

  View performance metrics for agents

  Manage users and complaints

- **Technical Stack Integration**

Frontend: React.js with Material UI & Bootstrap for responsive design

Backend: Express.js for server-side logic

Database: MongoDB for storing users, complaints, messages, etc.

Real-Time Communication: Socket.IO for live updates and chat

APIs: RESTful architecture with Axios for client-server communication

3.Architecture

- **Frontend :**

- **Role-Based Component Organization**

The frontend architecture follows a well-structured, role-based approach by organizing components into separate folders for each user type: **admin**, **agent**, and **user**. The admin folder includes components like AdminHome.jsx, AccordionAdmin.jsx, UserInfo.jsx, and AgentInfo.jsx, which are used to monitor platform activity, view complaints, and manage user/agent information. Similarly, the agent folder contains AgentHome.jsx, which is the central dashboard for agents to view and manage complaints assigned to them. The user folder holds components like Complaint.jsx, HomePage.jsx, and Status.jsx, which allow users to file

new complaints, view their dashboard, and track the status of submitted complaints. This separation ensures a clean, scalable design and easier maintenance as the application grows.

- **Shared and Common Components**

The common folder holds shared components that are reused across different roles in the application. This includes components like Login.jsx and SignUp.jsx for user authentication, Home.jsx which likely acts as the public landing or welcome page, and FooterC.jsx for the footer layout that appears across pages. A particularly important component here is ChatWindow.jsx, which facilitates real-time messaging between users and agents. By isolating these shared components, the architecture promotes **code reusability** and a **consistent user experience**, while preventing code duplication across different user types.

- **Central Application Files**

At the root of the src folder, core files like App.js, index.js, and App.css serve as the backbone of the application. App.js is the main component where routing is handled, likely using react-router-dom. It defines different routes and layouts based on the user's role, such as rendering AdminHome for admins and HomePage for users. App.css contains the global styles that apply across the entire app. The index.js file serves as the **entry point** of the React application. It imports App.js and renders it into the root div inside index.html, effectively bootstrapping the app.

- **Image and Static Asset Handling**

The Images folder stores static resources used throughout the application. For example, Image1.png may be a logo, banner, or an illustration used in UI components like headers or welcome pages. By storing assets in a centralized folder, the project maintains a clean and organized structure that makes it easy to manage and update visual elements without cluttering component logic.

- **Modular and Maintainable Design**

This architecture adopts React's **component-based** philosophy, meaning each .jsx file represents a self-contained unit responsible for a specific part of the UI. This modularity allows developers to isolate and focus on small parts of the app independently, making debugging, testing, and future feature additions much easier. The separation of components by user role also simplifies conditional rendering and role-based access, ensuring that each user sees only the UI relevant to them.

- **Scalability and Extensibility**

The current structure is well-prepared for future scalability. If the application needs to introduce new roles like "manager" or "super admin," it would be straightforward to add new folders and components without disrupting the existing ones. Similarly, new features such as notifications, dashboards, or reports can be developed in isolated modules and integrated seamlessly. This architectural design supports extensibility without compromising on readability or performance.

- **Backend :**

- **Express.js as the Server Framework**

The backend is built using **Express.js**, a minimalist Node.js framework that handles HTTP requests and defines API routes for communication between the frontend and the database. Express provides the structure to organize endpoints for functionalities such as user registration, complaint submission, message exchange, and role-based access. The modular route handling approach allows you to separate logic for different operations — for example, routes for complaints, messages, users, and admin actions — ensuring clean and maintainable code. Middleware like express.json() is used to parse incoming JSON data, making it easier to handle request bodies sent from the React frontend.

- **RESTful API Design**

The system follows a **RESTful API architecture**, which structures backend endpoints around resources such as /users, /complaints, /messages, etc. Each route is defined using appropriate HTTP methods (e.g., POST for creation, GET for retrieval, PUT for updates, and DELETE for deletions). For example, a POST /complaints route is used when a user submits a new complaint, and a GET /complaints/:id is used to fetch specific complaint details. These APIs are consumed by the React frontend using Axios, allowing seamless interaction between the client and server.

- **Node.js Runtime Environment**

The backend of ResolveNow is built on **Node.js**, a JavaScript runtime that enables asynchronous, non-blocking I/O operations—ideal for handling multiple user requests efficiently. Node.js allows developers to use JavaScript on the server side, ensuring consistency between frontend and backend logic. Its event-driven architecture is particularly well-suited for real-time applications like complaint tracking and agent-user chat, where continuous server communication is needed.

- **Role-Based Access and Middleware**

To ensure secure access and proper role-based functionality, the backend uses **middleware** for authentication and authorization. Middleware functions check if the incoming requests include a valid token (usually a JWT), and if the user making the request has the appropriate role (admin, agent, or user). This ensures that sensitive operations, such as assigning complaints or viewing all complaint records, are restricted to authorized users only. This approach enforces **security and access control** at the backend level.

- **Server Configuration and Environment Setup**

The Express server runs on a designated port (e.g., 8000) and connects to MongoDB using a connection string stored in environment variables. A .env file typically holds sensitive credentials like the database URI, JWT secret, and server port. This setup enhances **security and portability**, allowing the backend to run in different environments (development, testing, production) with minimal configuration changes.

Database :

- **MongoDB as the Database**

The backend uses **MongoDB**, a NoSQL database, to store structured collections of data such as user profiles, complaint records, message logs, and assignment data. Each type of data is represented as a collection with flexible JSON-like documents. MongoDB's schema-less nature makes it ideal for storing dynamic data such as user-submitted complaint forms, which may contain varying numbers of fields or attachments. The MongoDB collections are accessed using Mongoose, an ODM (Object Data Modeling) library that simplifies database queries, schema definition, and model validation.

- **Mongoose for Object Modeling**

The application uses **Mongoose**, an Object Data Modeling (ODM) library for MongoDB and Node.js. Mongoose provides a schema-based solution for modeling application data. It allows developers to define clear structures for collections (such as users, complaints, messages) and handle validation, data transformation, and complex queries in a clean and organized way. Mongoose also simplifies interaction with MongoDB by offering built-in methods for create, read, update, and delete (CRUD) operations.

- **User Schema**

The **User schema** defines the structure of user profiles within the application. Fields typically include name, email, password, role (e.g., user, agent, admin), and timestamps. Passwords are securely hashed before being saved. The role field plays a crucial role in authorization and interface rendering. Additionally, the schema may include fields like isVerified for email confirmation and contactDetails for future integrations (e.g., OTP, SMS alerts).

- **Complaint Schema**

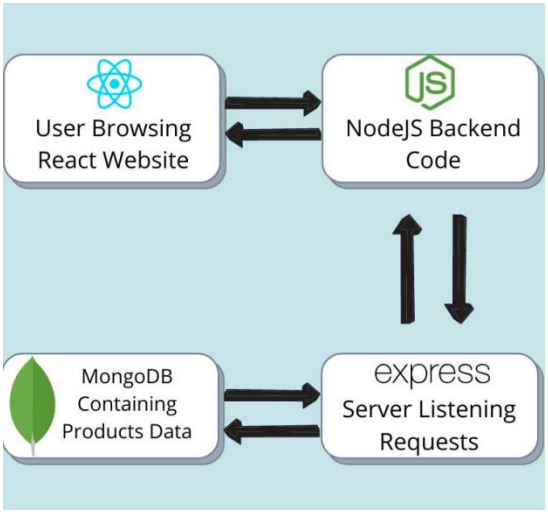
The **Complaint schema** is central to the platform. It includes fields such as title, description, status (e.g., pending, in-progress, resolved), userId (referencing the User), assignedTo (referencing the Agent), submissionDate, and optional attachments (images/documents). This schema enables the platform to track the lifecycle of each complaint from submission to resolution. Status changes are updated as the complaint moves through different stages, and references help manage relationships with users and agents efficiently.

- **Message Schema (Chat Functionality)**

For the chat feature, a **Message schema** is used to store real-time conversations between users and agents. Each message document contains the senderName, messageContent, timestamp, and a reference to the complaintId it belongs to. This linkage ensures that every conversation is tied to a specific complaint, allowing users and agents to view the entire discussion contextually. The schema supports scalability for multiple messages per complaint and can include indicators like isRead or messageType (text, image, etc.) if needed.

- **CRUD Operations and Data Access**

Interaction with MongoDB is performed through Mongoose methods like Model.find(), Model.findById(), Model.save(), and Model.updateOne(). These operations are encapsulated within route handlers or controller functions in Express. For example, when a user submits a complaint, a new document is created in the Complaints collection using ComplaintModel.create(). Similarly, when an agent updates the status of a complaint, a ComplaintModel.updateOne() operation is used. MongoDB's flexible schema and Mongoose's abstraction simplify these interactions significantly



4.Setup Instructions

- **Prerequisites :**

Here are the key prerequisites for developing a full-stack application using Node.js, Express.js, MongoDB, React.js:

- **Node.js and npm:**

Node.js is a powerful JavaScript runtime environment that allows you to run JavaScript code on the server-side. It provides a scalable and efficient platform for building network applications.
Install Node.js and npm on your development machine, as they are required to run JavaScript on the server-side.
Download: https://nodejs.org/en/download/
Installation instructions: https://nodejs.org/en/download/package-manager/

- **Express.js:**

Express.js is a fast and minimalist web application framework for Node.js. It simplifies the process of creating robust APIs and web applications, offering features like routing, middleware support, and modular architecture.
Install Express.js, a web application framework for Node.js, which handles server-side routing, middleware, and API development.
Installation: Open your command prompt or terminal and run the following command:

npm install express

- **MongoDB:**

MongoDB is a flexible and scalable NoSQL database that stores data in a JSON-like format. It provides high performance, horizontal scalability, and seamless integration with Node.js, making it ideal for handling large amounts of structured and unstructured data.

Set up a MongoDB database to store your application's data.
Download: https://www.mongodb.com/try/download/community
Installation instructions: https://docs.mongodb.com/manual/installation/

- **React.js:**

React.js is a popular JavaScript library for building user interfaces. It enables developers to create interactive and reusable UI components, making it easier to build dynamic and responsive web applications.
Install React.js, a JavaScript library for building user interfaces.
Follow the installation guide: https://reactjs.org/docs/create-a-new-react-app.html

- **HTML, CSS, and JavaScript**: Basic knowledge of HTML for creating the structure of your app, CSS for styling, and JavaScript for client-side interactivity is essential.

- **Database Connectivity**: Use a MongoDB driver or an Object-Document Mapping (ODM) library like Mongoose to connect your Node.js server with the MongoDB database and perform CRUD (Create, Read, Update, Delete) operations. To Connect the Database with Node JS go through the below provided link:
  https://www.section.io/engineering-education/nodejs- mongoosejs-mongodb/

- **Front-end Framework**: Utilize Reactjs to build the user-facing part of the application, including entering complaints, status of the complaints, and user interfaces for the admin dashboard.

For making better UI we have also used some libraries like material UI and boostrap.

- **Version Control**: Use Git for version control, enabling collaboration and tracking changes throughout the development process. Platforms like GitHub or Bitbucket can host your repository.

  Git: Download and installation instructions can be found at: https://git-scm.com/downloads

- **Installation :**

- **Development Environment**: Choose a code editor or Integrated Development Environment (IDE) that suits your preferences, such as Visual Studio Code, Sublime Text, or WebStorm.

  Visual Studio Code: Download from https://code.visualstudio.com/download

  To run the existing Video Conference App project downloaded from GitHub:
  Follow below steps:
  Clone the Repository:
  - Open your terminal or command prompt.
  - Navigate to the directory where you want to store the e-commerce app.
  - Execute the following command to clone the repository:

  git clone: https://github.com/awdhesh-student/complaint-registery.git

- **Install Dependencies:**

  - Navigate into the cloned repository directory:
    cd complaint-registery
  - Install the required dependencies by running the following commands:
    cd frontend
    npm install
    cd ../backend
    npm install
- **Start the Development Server:**
  - To start the development server, execute the following command:
    npm start
  - The online complaint registration and management app will be accessible at http://localhost:3000

You have successfully installed and set up the online complaint registration and management app on your local machine. You can now proceed with further customization, development, and testing as needed.

5.Folder Structure

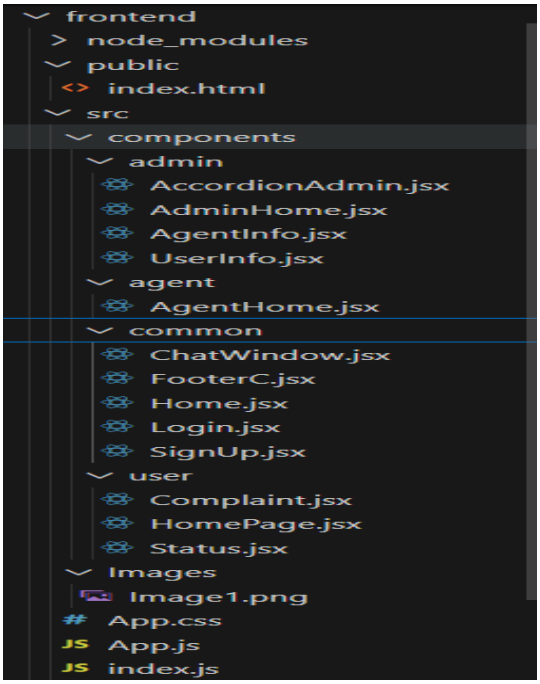
- **Client :**

The frontend of the **ResolveNow** complaint management system is developed using **React.js** and follows a modular, component-based architecture to ensure scalability, maintainability, and clean separation of concerns. At the heart of the frontend lies the src/ directory, which houses all the JavaScript and JSX code. Within it, the components/ folder is organized into subdirectories based on user roles (admin, agent, and user) and common elements (common). This clear division allows each user type to have a tailored interface while reusing shared logic and layout components.

The admin/ folder contains components like AdminHome.jsx, AgentInfo.jsx, and UserInfo.jsx, responsible for administrative operations such as managing users and overseeing complaint assignments. It also includes AccordionAdmin.jsx, likely used to present dynamic lists of complaints or agent tasks in a collapsible UI. The agent/ folder includes AgentHome.jsx, which serves as the dashboard for agents to view and resolve complaints assigned to them. The user/ folder contains components such as Complaint.jsx, HomePage.jsx, and Status.jsx, which facilitate complaint submission, status tracking, and user-specific interactions.

The common/ folder hosts components used across all roles, including ChatWindow.jsx for real-time chat between users and agents, Login.jsx and SignUp.jsx for authentication, Home.jsx for the landing page, and FooterC.jsx for the site footer. The presence of these shared components promotes reusability and consistency in design across the application. Additionally, the Images/ folder stores visual assets like banners or logos used in the UI.

The application is bootstrapped using index.js, the entry point of the React app, which mounts the root component (App.js) into the HTML DOM. The App.js file manages the routing between different components using React Router, directing users to the appropriate views based on their authentication status and role. Styling is handled globally through App.css, ensuring consistent design and responsiveness. Overall, this frontend structure enables a clean, user-centric interface that can be easily extended as the platform evolves.



- **Server :**

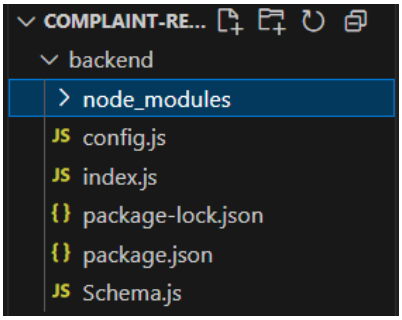
The backend of the **ResolveNow** complaint registration and management system is built using **Node.js** with the **Express.js** framework, following a minimal yet effective project structure that supports maintainability and scalability. At the top level, the backend/ folder contains the core server-side files required to run the application, connect to the database, and expose RESTful APIs.

The entry point of the application is index.js, which serves as the primary script that initializes the Express server, configures middleware such as express.json() and cors(), and defines routes for handling API requests. It also connects to the MongoDB database and starts the server on a specified port. The config.js file is responsible for centralizing important configurations such as the MongoDB connection setup using Mongoose. This separation ensures that database logic is modular and easily manageable.

The Schema.js file plays a critical role by defining the **Mongoose schemas and models** for MongoDB. It includes the data structure for key entities like User, Complaint, AssignedComplaint, and Message, allowing the application to perform structured CRUD operations on these collections. All data interactions within the app, such as complaint registration, status updates, and user management, rely on these models.

Additionally, the package.json file holds metadata about the project, including its dependencies (express, mongoose, cors, etc.), scripts, and environment requirements. The package-lock.json file ensures consistent dependency versions across environments. The node_modules/ directory contains all the installed packages required to run the backend server.

This lightweight yet well-organized backend structure effectively supports REST API development, MongoDB integration, and real-time interactions, making it easy to expand or refactor the system as new features are added.



6.Running the Application :

In the directory were folder is placed open in visual studio code open two terminals run the frontend and backend simultaneously

- **Frontend :**

  cd complaint-registery/frontend

  npm install # (only required once to install dependencies)

  npm start

  This will launch React development Server at :

  http://localhost:3000

- **Backend :**

  cd complaint-registery/backend

  npm install # (only required once to install dependencies)

  npm start

  This will start the Express Server at :

  http://localhost:8000

7.API Documentation :

- **User Login**

The /Login endpoint allows users to authenticate themselves by submitting their email and password via a POST request. This route verifies the user credentials against the database. If valid, it returns the user details

including ID, name, and user type. If credentials are invalid or the user doesn't exist, appropriate error messages are returned. This functionality is available to all users of the system.

- **Request**

email: "ara@gmail.com"
password: "123@ara"

- **Response**

_id: ObjectId('685c0ed13137328c0766d71b')
name: "MOHAMMAD AARA"
email: "ara@gmail.com"
password: "123@ara"
phone: 89293472863
userType: "Ordinary"
createdAt: 2025-06-25T14:59:29.094+00:00
updatedAt: 2025-06-26T02:39:25.516+00:00
__v: 0

By using the data it verifies the user email and password to login

- **User Registration**

The /SignUp endpoint enables new users to register by providing their name, email, password, phone number, and user type. It accepts a POST request and stores the user information in the database. On successful registration, the user's ID, name, and email are returned. If an error occurs during the save operation, the server responds with a 500 Internal Server Error. This is a public endpoint accessible to anyone who wants to sign up.

- Request

_id: ObjectId('685c0ed13137328c0766d71b')
name: "MOHAMMAD AARA"
email: "ara@gmail.com"
password: "123@ara"
phone: 89293472863
userType: "Ordinary"
createdAt: 2025-06-25T14:59:29.094+00:00
updatedAt: 2025-06-26T02:39:25.516+00:00
__v: 0

- Response

_id: ObjectId('685c0ed13137328c0766d71b')
email: "ara@gmail.com"
password: "123@ara"

- **Submit a Complaint**

The /Complaint/:id endpoint is used by ordinary users to submit complaints. A POST request is made with the complaint details such as name, city, state, pincode, address, comment, and status. The server uses the provided user ID to associate the complaint with the correct user. On successful registration, the server returns the complaint ID and status. If the user ID is not found, a 404 Not Found error is returned.

- **Request**

```
_id: ObjectId('685c16483137328c0766d729')
userId: ObjectId('685c0ed13137328c0766d71b')
name: "Manntasha Aara"
address: "Naizampeta , gudivada"
city: "Gudivada"
state: "Andhra Pradesh"
pincode: 521301
comment: "electricity"
status: "pending"
__v: 0
```

- **Response**

```
_id: ObjectId('685c16483137328c0766d729')
userId: ObjectId('685c0ed13137328c0766d71b')
status: "pending"
```

- **Get All Complaints (Admin)**

Admins can retrieve all registered complaints using the /status endpoint through a GET request. This returns a list of all complaints along with their IDs, names, and statuses. It's useful for monitoring the volume and nature of complaints in the system. If an error occurs during data retrieval, a 500 Internal Server Error is returned. Access to this endpoint is restricted to admin users only.

- **Response**

```
_id: ObjectId('685c16483137328c0766d729')
comment: "electricity"
status: "pending"
```

- **Assigned Complaints (Admin)**

The /assignedComplaints endpoint allows admins to assign complaints to agents via a POST request. The request body must include the agent ID and complaint ID. Upon successful assignment, the server returns a confirmation message. If there is a problem with saving the assignment, the server responds with a 500 Internal Server Error. This endpoint is restricted to admin users.

- **Request**

```
"_id": {
    "$oid": "685c10553137328c0766d71f"
  },
  "userId": {
    "$oid": "685c0ed13137328c0766d71b"
  },
```

- **Response**

It displays the message

- **Agent Complaint List**

Agents can retrieve a list of complaints assigned to them using the /allcomplaints/:agentId endpoint. A GET request is made using the agent's ID as a URL parameter. The server responds with an array of complaint details such as name, city, and status. If there is a server-side error while fetching data, a 500 Internal Server Error is returned. This route is accessible only to agents.

- **Response**


```
_id: ObjectId('685c3b76b46bb25f43892be3')
agentId: ObjectId('685c3b3bb46bb25f43892bd6')
complaintId: ObjectId('685c16483137328c0766d729')
status: "pending"
agentName:  "David Sukumar"
```


- **Update Complaint Status (Agent)**

Agents can update the status of a complaint using the /complaint/:complaintId endpoint. This is done via a PUT request with the new status in the request body. The server locates the complaint using the provided ID and updates its status. If the complaint ID is missing or invalid, the server responds with a 404 Not Found error. This functionality is designed for agent-level users.

- **Request**


```
status: "completed"
```


- **Response**


```
complaintId: ObjectId('685c16483137328c0766d729')
status: "completed"
```


- **Cancel Complaint**

The /complaints/:id/cancel endpoint allows both admins and agents to cancel a complaint. A PUT request is sent with the complaint ID in the URL. The system updates the complaint status to “Cancelled” and also updates or deletes related assignment records accordingly. A success message is returned upon successful update. If the operation fails, a 500 Internal Server Error is returned. This route ensures that invalid or redundant complaints are effectively managed.

- **Response**


```
_id: ObjectId('685c16483137328c0766d729')
userId: ObjectId('685c0ed13137328c0766d71b')
status: "cancelled"
__v: 0
```


8.Authentication

The authentication mechanism in the ResolveNow project is implemented using a simple login strategy where the user submits an email and password combination. Upon successful login, the backend validates the credentials using MongoDB and Mongoose. If the user exists and the credentials match, the backend returns the full user object, including the user’s role (such as Admin, Agent, or Ordinary). This user data is then stored on the frontend using localStorage, which keeps the user logged in and allows the application to persist user sessions across page reloads.

Authorization is handled entirely on the client side by checking the userType stored in localStorage. This value determines which components and routes are accessible to the user. For example, only users with userType: "Admin" can access admin-specific pages such as assigning complaints or managing users. Similarly, agents are directed to their dashboard where they handle assigned complaints, and ordinary users can only register and track their complaints.

For future improvements, it is recommended to implement JWT-based token handling or session-based authentication using libraries such as express-session. These additions would allow secure, stateless user authentication, proper route protection on the backend, and more reliable session handling across different platforms and devices.

9. User Interface :

- **Technologies Used:**
  - **React.js**
  - **Material UI**
  - **Bootstrap**
  - **Ant Design**
  - **React-Bootstrap**
  - **Axios (for API calls)**

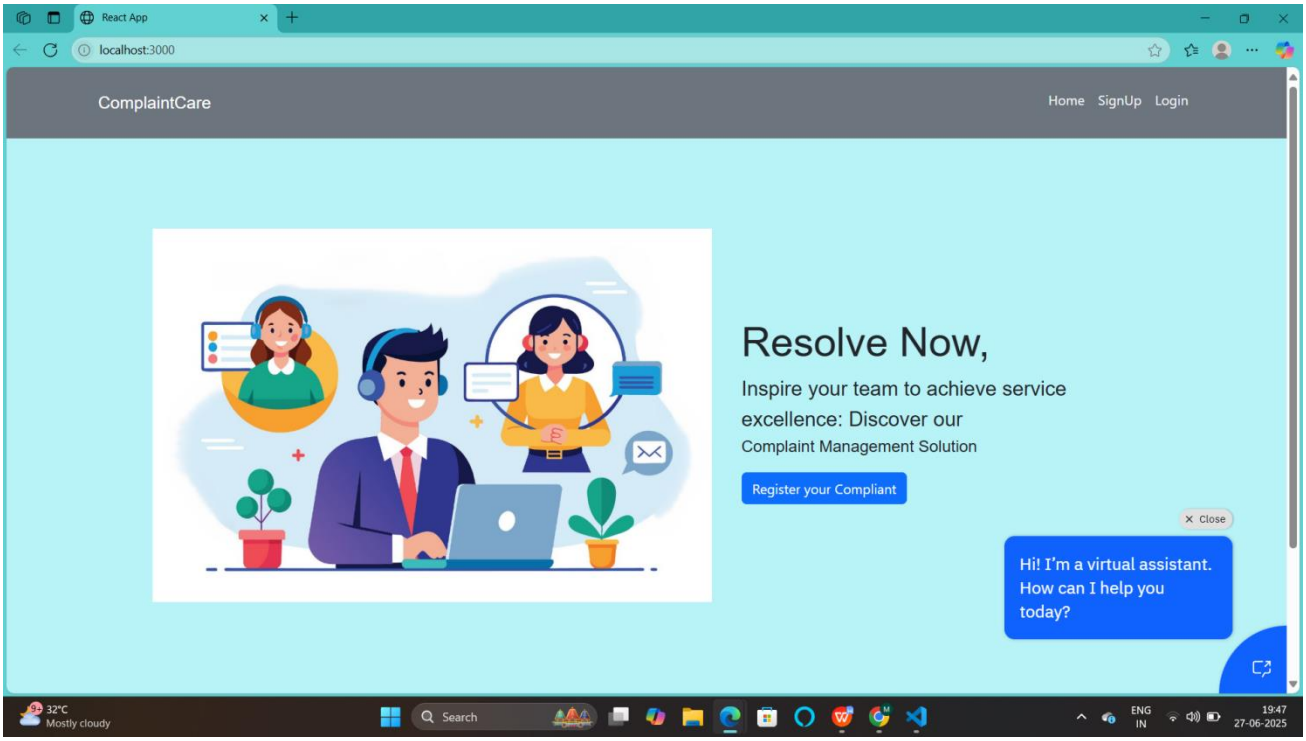
- **Main Pages:**
  - **Landing Page**:
    Displays a welcome message with options to log in or register. It acts as the entry point for all types of users (Ordinary, Agent, Admin).
  - **Login/Register Page**:
    Provides authentication forms for new user registration and existing user login. Based on credentials, users are redirected to role-specific dashboards.
  - **Dashboard (Role-Based)**:
  - **Ordinary User View**: Allows users to submit new complaints, view status updates, and track previous complaints.
  - **Agent View**: Displays complaints assigned to the agent, with options to update complaint status and communicate with users.
  - **Admin View**: Enables admins to assign complaints to agents, manage all users, delete users, and monitor platform-wide complaint activity.
  - **Complaint Detail Page**:
    Shows complete complaint details submitted by the user including the comment, location, and current status. Allows agents/admins to view and take action on specific complaints.
  - **Complaint Status Page**:
    Lists all complaints submitted by the logged-in user with status labels such as Pending, Assigned, Resolved, or Cancelled. Helps users keep track of progress and actions taken.

10.Testing :

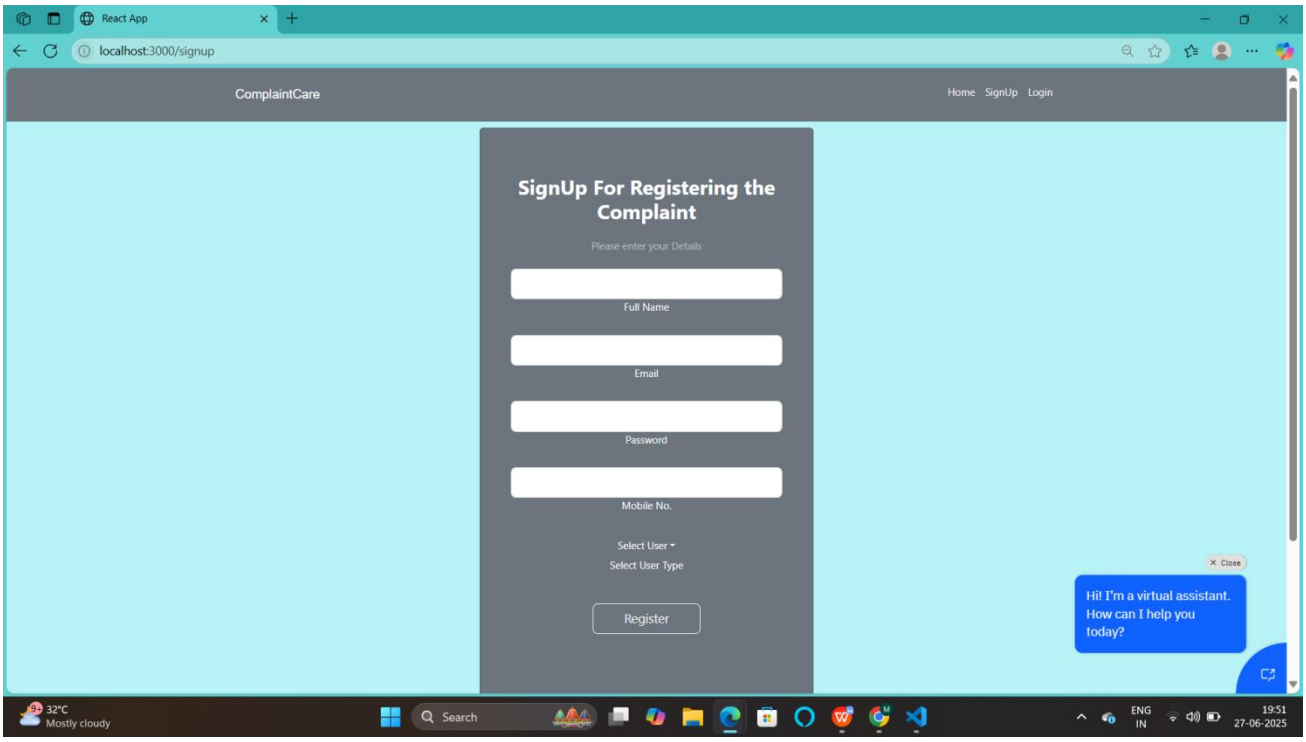
- **Manual Testing:**
  Frontend: Via browser – test property listings, booking forms, dashboard functionalities.
  Backend: Use **Postman** to manually test each API.
- **Automated Testing:**
  Backend: Use **Jest** and **Supertest** for API testing.
  Frontend: Use **React Testing Library** and **Jest** to test UI components and API integration

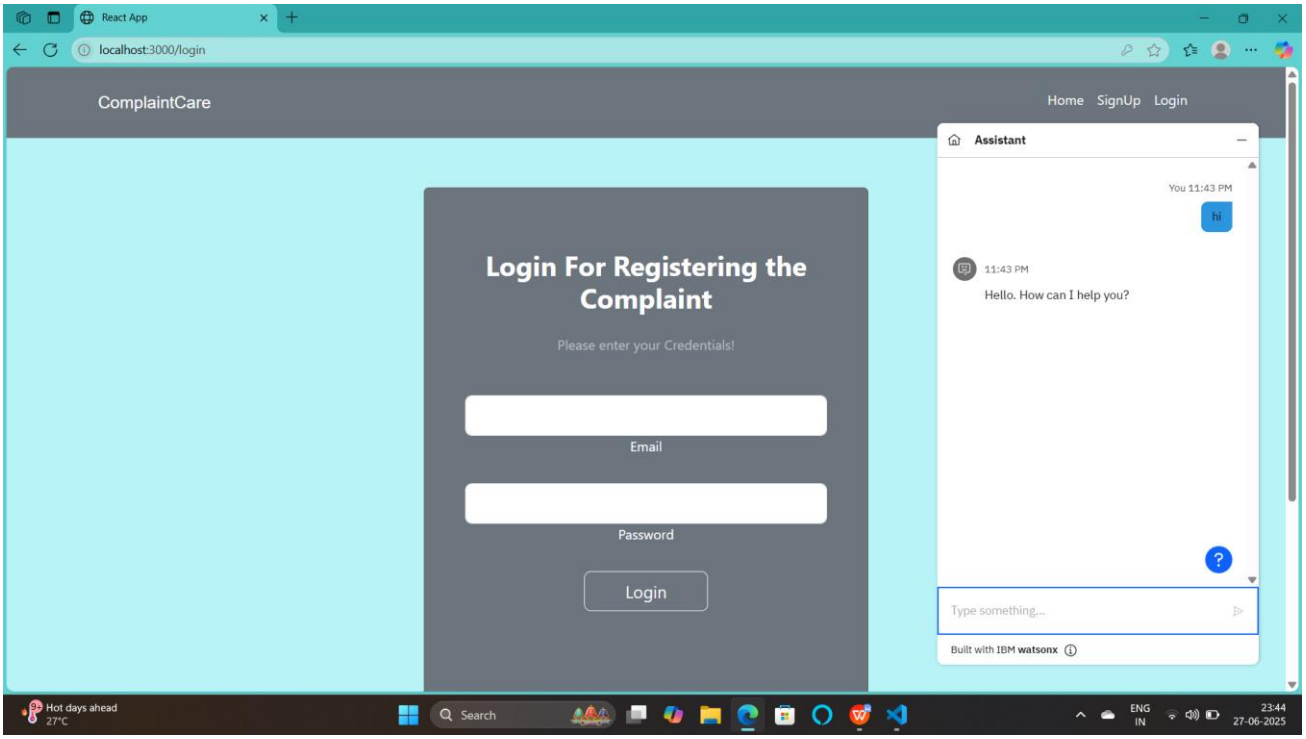
11.Screenshots

- **Landing Page**

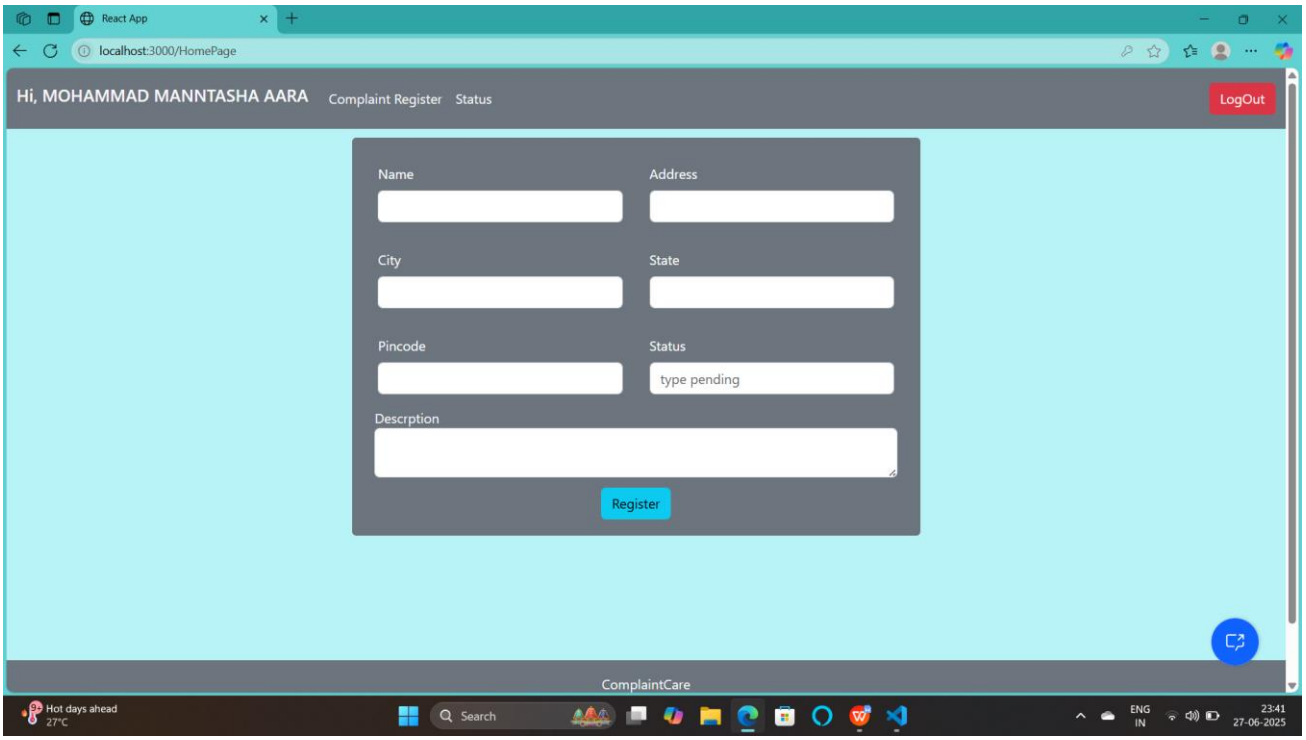


- **Registration and Login Page**

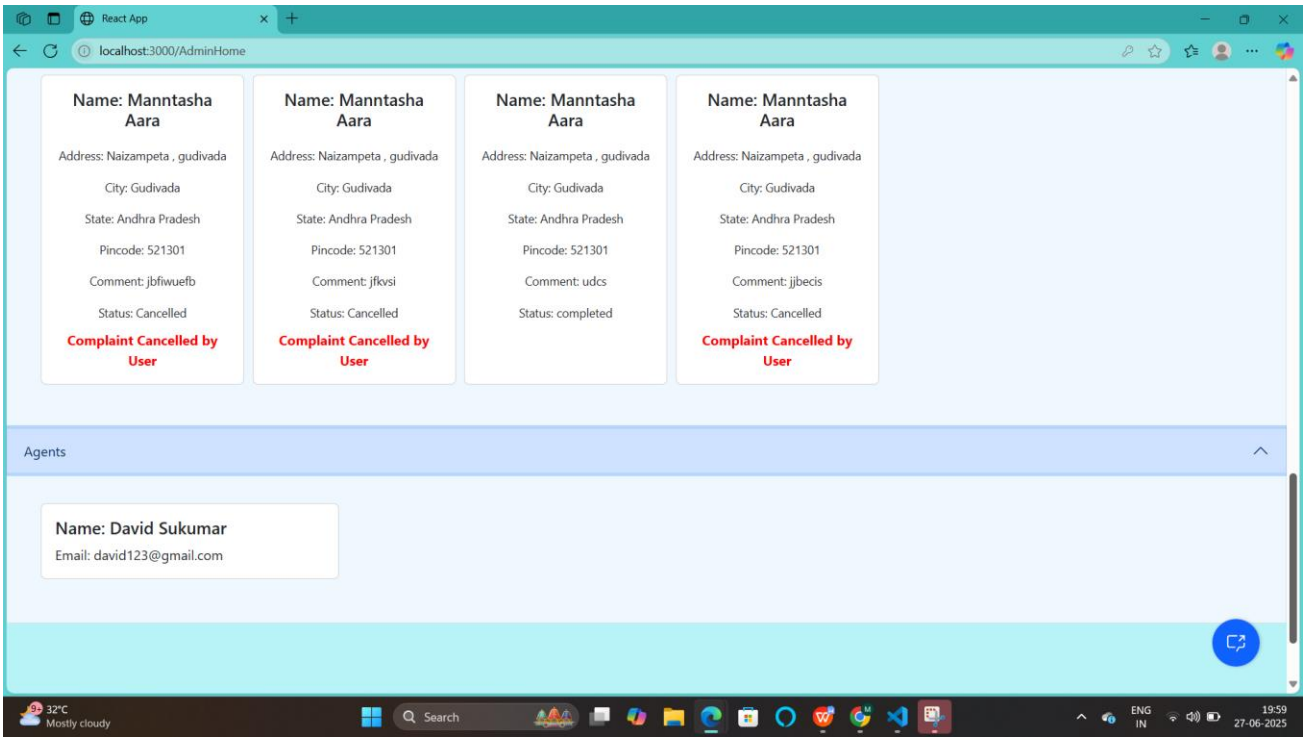




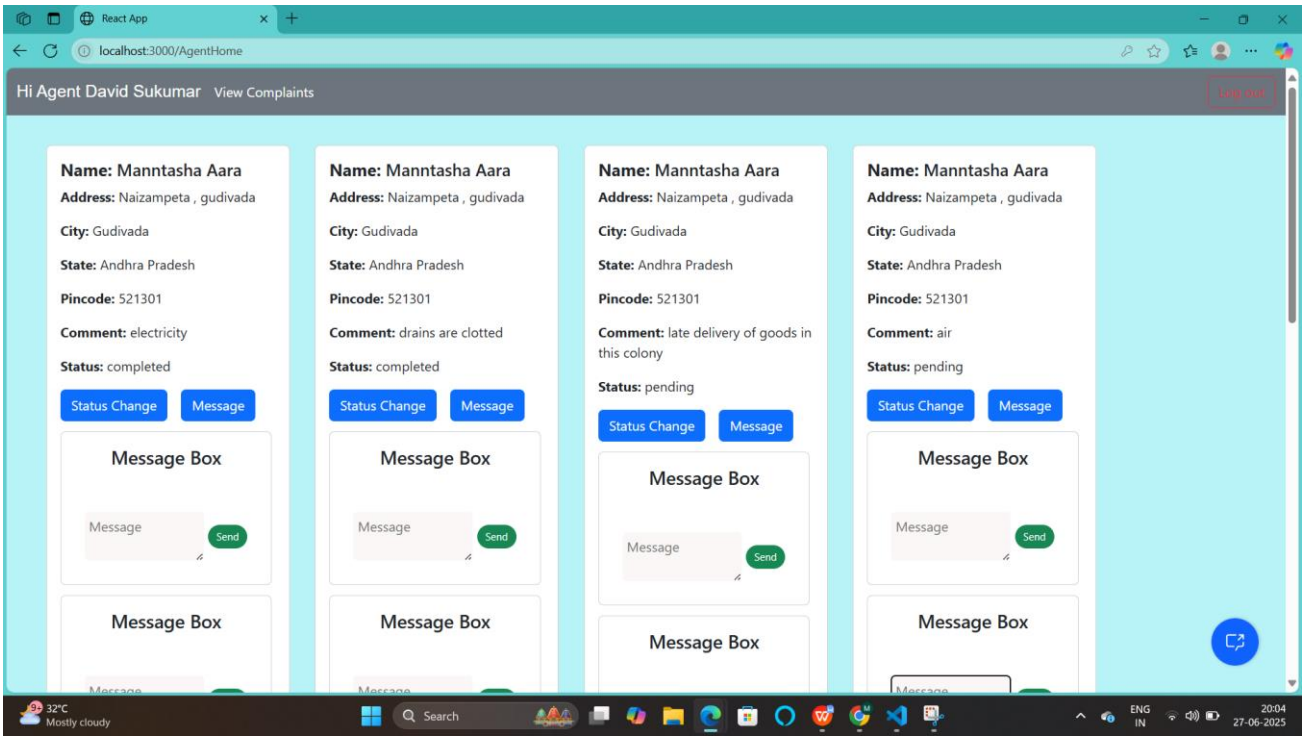
- **Home Page**
  - **User**



- **Admin**



- **Agent**



12..Known Issues

- **No Token-Based Authentication**:
  Currently, the system uses basic email-password login without any token mechanism like JWT. This makes the app vulnerable to unauthorized access, as all authentication and session control is handled solely through localStorage, which is not secure for sensitive data.
- **No Real-Time Updates**:
  Complaint status changes and agent-user communication do not update in real-time. Users have to refresh the page manually to see new messages or updates.

- **Frontend Session Persistence Only via LocalStorage**:
  LocalStorage is used for session tracking, which is easily accessible from browser dev tools. If a user modifies the userType, they could gain unauthorized access to protected components.
- **No Pagination or Filtering for Admin Dashboard**:
  All complaints and users are shown in a single view, which becomes difficult to manage as the dataset grows.

13. Future Enhancements

- **Image/File Upload for Complaints**:
  Allow users to **attach screenshots or documents** when submitting complaints, using Multer or similar middleware for file handling.
- **Email and SMS Notifications**:
  Integrate services like **Nodemailer** or **Twilio** to send updates and confirmations to users when complaints are registered, updated, or resolved.
- **Analytics & Reporting**:
  Build a module for **visual analytics** (charts, graphs) to help admins track complaint trends, resolution times, and agent performance.
- **Progressive Web App (PWA) Support**:
  Make the system installable on mobile devices by converting it into a **PWA** for better accessibility and offline support.